Deep Space Network

Multi-Use Software

Year 2000 Code Inspection:

Final Report

Sept 22, 1997

Prepared by: Van Hoang - MSW S/W Developer
Reviewed by: Steve Rockwell - MSW CDE

ISDS Team

Prepared for:

JET PROPULSION LABORATORY
PASADENA, CALIFORNIA
Contract No. 960100

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

1.1 Identification

This document presents findings and recommendations resulting from the Y2K code inspection of the DSN Multi-Use Software (MSW) Program set DOI-5555-OP-A in accordance with DSN Year 2000 Compliance Requirements document. Code inspection is the first step in the Y2K compliance process. Later steps involve validation and demonstration of a program set through testing.

1.2 Overview

In general, MSW not only constructs frameworks for DSN subsystem/assembly application software to build upon, it also provides a diverse set of Application Programming Interfaces (APIs) and tools to realize and facilitate the implementation of all other DSN subsystems. Under inspection is MSW version 1.8.2 which consists of 165 directories with 1608 files and supports Solaris, VxWorks, VadsWorks, Realix, OS/2, and Power PC platforms.

1.3 Scope

This document specifies approach, dependencies, and analysis employed during the code inspection process. It also spells out potential anomalies and listing of MSW time-related functions and variables for use in subsequent code inspection of other DSN applications built upon MSW.

1.4 Controlling Documents and References

820-055 DSN Year 2000 Compliance Requirements, May 22, 1997.
829-021 DSN Year 2000 Compliance Test Document, Draft, September 8, 1997.
UG-DOI-5555-OP-A User's Guide - Mutil-Use Software, November 1994.

DSN Year 2000 Compliance Web Page, URL:
http://deepspace1.jpl.nasa.gov/940/private/year2000

ISO 8601:1988 Date/Time Representations
available from ftp.informatik.uni-erlangen.de/pub/doc/ISO/ISO8601.ps.Z

http://newproducts.jpl.nasa.gov/travel/technqs.pdf (Excerpt from IBM's "The Year 2000 and 2-Digit Dates")

http://www.RighTime.com/pub/year2000.txt

The Single UNIX Specification, Version 2, 1997 of The Open Group

JPL Year 2000 Web Page, URL:
http://newproducts.jpl.nasa.gov/forms/jplyr2k1.htm

SECTION 2

MSW Y2K CODE INSPECTION APPROACH

2.1 Emphasis

Although this task calls for inspection of all code, the following criteriawere emphasized and used to determine which files/modules should be reviewed first with utmost attention:

- Usage of ANSI/POSIX time-related library functions such as ascftime, asctime, asctime_r, cftime, ctime, ctime_r, difftime, getdate, gettimeofday, gmtime, gmtime_r, localtime, localtime_r, mktime, settimeofday, strftime, strptime, time, txset, tzsetwall, utime, DosGetDateTime, timex, tickAnnounce, tickGet, tickSet, taskDelay, dosFsDateSet, or rt11FsDateSet.

- Logic and variable size used in date and time comparison, conversion, manipulation, leap year determination, handling of leap second, and user interfaces.

- Understanding expected behavior of time-related functions.

2.2 Process

MSW files were reviewed through different passes with file lists resulting from a search for a set of keywords.

- Pass 1: Review modules with calls to time-related standard library functions. Identify timevariables and time-related MSW functions (used as keywords for pass 2).

- Pass 2: Review MSW-provided time functions for Y2K compliance. Provide notes on expected behavior as compared to User's Guide.

- Pass 3: Review remaining files.

- Pass 4: Simple stand-alone tests using existing Auto Control (ACTL) Scripts.

2.3 Dependencies

Determination of Y2K compliance for MSW also depends on the compliance of underlined operating systems and C standard library functions. It is necessary to assume that these dependencies are Y2K compliant until they can be validated and demonstrated in step 2 of the compliance process.

SECTION 3

MSW Y2K CODE INSPECTION DELIVERABLES

3.1 Anomalies

The following anomalies were detected and documented using DSN Anomaly Reporting System (ARS):

- **AR 30466** - Category B, Priority 2:
  Description: *MSW **Get_time** function provides current time based on user's choice of TCT time, System Time or Best Time. The anomaly occurs when user requests System Time AND the seconds of year is greater than seconds in year. Under this condition (at last second of a year with leap second), **Get_time** sets 4-digit year to 19xx instead of 20xx.*

- **AR 30467** - Category B, Priority 2:
  Description: *MSW **tct_btime** (Sun Version) returns TCT time and TCT status flag. However, if TCT error occurs, **tct_btime** returns incorrectly computed day-of-year for February and later month.*

- **AR 30468**- Category C, Priority 3:
  Description: *MSW **CSmtime** (Sun and Realix version) returns current time in milliseconds (since 1970-01-01 00:00:00 UTC). It wouldresult in receiving variable (unsigned long) over-flow. However, consecutive calls to derive time difference yield correct result.*

3.2 MSW Time-Related Function and Variables

It is essential to understand expected behavior of MSW functions and data structures for proper usage. Appendix A spells out MSW provided time-related functions and associated data structures with its Y2K compliant status. For convenience, the following list of keywords can also be used in a search tool to locate lines of code related to time:

- MSW time-related function name:
  CSwait, CSsuspend_tak, CSmtime, Get_time, Get_gmt, Get_syr, Set_year, Set_time_bias, Elapse, Correct_seconds_of_year, Sec_per_year, Time_diff, csdoytim, Tctime, tct_btime, set_time_vals, tct_time, CSbtcvt, CSmdtime, CStctmdt, CStct2bin, CSdsptime, CSops68, cvt_bin_to_tct_string, cvt_tct_string_ to_bin, gtime, Seconds_of_day_2_str, Seconds_of_year_2_str, Offset_of_year_2_str, Str_2_seconds_ of_day, Str_2_seconds_of_year, Str_2_offset_of_year.

- MSW time-related variables and definitions:
  TCT_TIME members: year, days_in_year, day, hr, min, sec, tenths, millisecs_of_day, sec_of_day, sec_of_year, time_bias; struct time_chunk, TIME_CHUNK members: time_bias, sys_itme, tct_status, tct_time, year, days_in_year, sec_per_year; struct tct_data members: status_byte, doy, sys_time_offset, tct_error; struct time_value members: days, hours, minutes, seconds, millisecs; struct btime_value members:
  bdays, bhours, bminutes, bseconds, bmillisecs, bmsecs_of_day; struct status_flags members: t_type, sub_leap_sec, add_leap_sec, leap_year; struct time_struct members: t_val, t_stat; DATETIME, struct _DATETIME members: hours, minutes, seconds, hundreds, day, month, year, timezone, weekday, struct tctm members: hour, mins, secs, mls; struct stat_list_node member: iv_doy_tag; RPT_CAT, struct rpt_catalog_block member: datetime, SEC_PER_DAY, SEC_PER_YEAR, SEC_PER_MIN, MSEC_PER_SEC, NSEC_PER_SEC, SEC_PER_YEAR, DAYS_IN_YR, DAYS_IN_LEAP_YEAR, ICMC_DAYS_PER_YEARNL, ICMC_DAYS_PER_YEARL, DAYS_PER_YEAR, LCMC_MS_PER_DAY, LEAP_MSEC_MOD, DAYSPERYR, HRSPERDAY, MAX_DOY_PER_YEAR.

3.3 MSW Date Representations

In general, MSW adheres to 820-016 DSN Subsystem Interface modules where applicable for data interchange. Therefore, this section is limited to listing of date formats used within MSW and as output to user's console:

- NSW software hard-coded version date: MM/DD/YY
- Program History (when auto-generated): MM/DD/YY
- Display and Report: DDD HH:MM:SS
- Error messages: DDD HH:MM:SS
- ACTL Log: DDD HH:MM:SS

SECTION 4

MSW Y2K CODE INSPECTION RECOMMENDATIONS

4.1 Analysis

It is likely that MSW is Y2K compliant at root. The reason is not only because MSW was developed and nurtured by best group of engineers whose expertise was the foundation of DSN implementation since MK-IVA through the SPC Upgrade era, but also because MSW deals mostly with day of year. The code inspection task, however, is still laborious and time consuming due to the following shortfalls:

- Lack of central test assertions for date and time functions:
  For example, the logic to determine leap year is scattered and different from one module to the other.

- Common interface but different behavior per platform:
  MSW CSmtime() is a good example of such implementation. Under Solaris and Realix, CSmtime returns milliseconds of current time (since UNIX time); where as, CSmtime in OS/2 returns current time in milliseconds relative to established time base; VxWorks returns milliseconds since start-up (of kernel's tick counter).

- Excessive definitions of an entity:
  For example, DAYS_IN_YR, ICMC_DAYS_PER_YEARNL, MAX_DOY_PER_YEAR, and DAYSPERYR all refer to 365 (days.)

4.2 Risk Assessment

Since the MSW code inspection process is based on certain assumptions and limited tool set, risk is an unavoidable fact of life. Nevertheless, it is important to identify known items as follows:

- Validity of C time-related Standard Library functions on different platforms:
  The occurrence of "Mar 00, 2000" problem in VxWorks calls for the need to validate time-related functions on all platforms. This effort will minimize risks associated with the original assumption.

- Fixed window technique:
  Y2K anomaly correction in MSW uses this technique to resolve dates with 2-digit year fields per {Y2K25} requirement. Exchange of 2-digit-year fields between MSW and other program therefore must use the same assumption. (Year value range 69-99 refers to the twentieth century and 00-68 to twenty-first century.)

- Simplified logic to determine leap year:
  Existing MSW code uses a simplified test of Modulo 4 (year % 4) to determine leap year. Since this logic works for DSN applicable year range of 1998 to 2015 {Y2K24}, further correction should be waived.

4.3 Recommendations

At best, the MSW Y2K code inspection task should provide some psychological assurance for users of MSW. Howeveer, subsequent Y2K validation and demonstration is the ultimate proof for Y2K compliance. Until then, please consider the following recommendations:

- Users of MSW should understand expected behavior of MSW-provided time-related functions and its associated data structure.

- MSW should provide SIM time capability to facilitate testing of the subsystem. With SIM time, MSW and application software can be tested for Y2K compliance without interfering with the operating system. Hence, a dedicated test bed may not be necessary, though recommended.

- MSW should provide common test assertions for date and time function. A common utility library for the handling and validation of date and time will minimize risks associated with the millennium rollover.

- MSW Y2K compliance does not shield other applications from millennium bugs. However, developers can minimize the risk by limiting usage of time-related functions to those provided by MSW.

- Time-related features provided by an Operating System should also be validated as soon as feasible. In-house validation of C standard library functions may be more cost effective and beneficial than relying on vendors' claim or researching news groups.

APPENDIX A

MSW TIME-RELATED FUNCTION INTERFACES

AND DATA STRUCTURES

This Appendix contains common time-related functions provided by MSW program set. Macro and constant definitions are included solely as keywords for further search of time-related lines of code. Also included are popular date/time-related structures along with members' definitions to avoid ambiguity.

A.1 MSW Interfaces

short CSwait(short timeout):
OK: suspend in msecs (up to 32000) or indef (-1 until signaled).

short CSsuspend_task(xsusiptr sp_susinfo):
OK: suspend task for specified secs or indef (0 until resumed).

ulong CSmtime() returns time in msecs. Sun and Realix returns current time. Other platform returns difference between current time and its own ref
ANOMALY: (Sun/Realix) msec since 01/01/70 00:00:00 UTC > ulong.

Get_time(short type, TCT_TIME *cur_time): Y2K_ANOMALY
gets current time and places it in the TCT_TIME structure.

Get_gmt(short type, short *day, long *sec):
OK: get current day and seconds of days

Get_syr(short type, long *secs):
OK: gets current seconds of year

Set_year(short year):
OK: updates TCT_TIME.year (YYYY) and days_in_year, sec_per_year.

Set_time_bias(long bias):
OK: updates TCT_TIME bias (Ssw_time->time_bias += bias).
Note that an adjustment is done instead of being set since the application will be calculating a bias adjustment from the system time that already includes existing bias.

long Elapse(long reftm):
OK: returns elapsed time (msecs) from provided time (msecs).

long Correct_seconds_of_year(long old_time):
OK: returns Ssw_time->sec_per_year + oldtime (if oldtime<0)
else returns old_time - Ssw_time->sec_per_year
long Sec_per_year():
OK: returns Ssw_time->sec_per_year.

short Time_diff(long a, long b): a/b time in secs of year
OK: returns -1, 0, or 1 for <, =, or >, respectively.

csdoytim(char *daytim):
OK: returns "DDD HH:MM:SS" of current time.

TCtime(struct time_value, struct status_flags):
OK: updates TCT time_value with GMT time (where TCT unavail).

tct_btime(struct btime_value *, status_flag *)
ANOMALY: (Sun) btime->days off by one month if set_sts_flags fails!

set_time_vals(long cur_time, int day, strct btime_value *btcts)
OK: set bmsecs_of_days with cur_time (in millisecs of day).

tct_time(struct time_value *tcts, struct status_flag *tctstat):
OK: sets tcts with current time (or TCT-adjusted ticks for Vadsworks)
ANOMALY for Sun version (cstctsn.c) since itcalls tct_btime above.

short CSbtcvt(long msecs, struct tctm *tcout)
OK: converts msecs to hrs, mins, secs and msecs.

short CSmdtime(short doy, long msecs, char *mdtime)
OK: converts doy & msecs to null-term string DDDHHMMSSsss.

short CStctmdt(struct time_value *tct, char *mdtime)
OK: converts time_value to null-term string DDDHHMMSSsss.

short CStct2bin(struct time_value *tct, short *doy, long msecs)
OK: converts time_value into doy and msecs (of day).

short CSdsptime(short doy, long msecs, cahr *dsptime)
OK: converts doy and msecs into null-term string "DDD HH:MM:SS"

long CSops68(long msecs)
OK: converts (returns) msecs to centi secs.

void cvt_bin_to_tct_string(const struct btime_value *btcts, struct time_value *tcts)
OK: converts btime_value to time_value.

cvt_tct_string_to_bin(const struct time_value *tcts, struct btime_value *btcts)
OK: converts time_value to btime_value.

static long gtime()
Establish and maintain a local time base, and return current time in msecs
(relative to the established time base - readjusted every 46+ days).
ANOMALY: consists of variables (ulong) whose length (32-bit) is
not adequate to handle time (msecs) since 1970-01-01 00:00:00 UTC.

Seconds_of_day_2_str(long time, char *time_fmt)
OK: Converts input secs (secs of day) to string blank/-HH:MM:SS
where minus sign is used for negattive input time.

Seconds_of_year_2_str(long time, char *time_fmt)
OK: Converts input secs (secs of year) to string blank/-DDD HH:MM:SS.

Offset_of_year_2_str(long time, char *offset_time)
OK: Converts time offset (secs of year) to string format
blank/[-]ddd HH:MM:SS where ddd indicates number of offset days.

Str_2_seconds_of_day(char *time_fmt, long *time)

OK: Converts string [+/-]HH[:]MM[:]SS to seconds (of year).

Str_2_seconds_of_year(char *time_fmt, long *time)
OK: Converts string [+/-][DDD]HH[:]MM[:]SS to seconds (of year).

Str_2_offset_of_year(char *offset_time, long *time)
OK: Converts string [+/-]ddd[ ]HH[:]MM[:]SS to offset time (seconds of year)

cbool ODistime(char *pval, union parm_vals *vout)
OK: Determines if the given value can be a time value.

A.2 Macros and Constant

Following are macros and constant definitions used within MSW. This listing is no more than a list of KEYWORDS for Y2K tools to locate time-related lines of code. Constant definition is not right/wrong by itself. Subsequent use of it, however, may result in Y2K non-compliance.

a. ssw/include/spc.h:

```
SEC_PER_DAY 86400
SEC_PER_HR      3600
SEC_PER_MIN     60
MSEC_PER_SEC    1000
NSEC_PER_SEC    1000000000
SEC_PER_YEAR            SEC_PER_DAY * 366
DAYS_IN_YR      365
DAYS_IN_LEAP_YR  366

CENTURY_NUM             1900 (Not used)
```

b. Local within ssw/lib/gettime.c:

```
DAYS_IN_YEAR( year )    (365 + ((0 == (year) % 4) ? 1 : 0))
```

c. csw/include/csw/csmd.h:

```
ICMC_DAYS_PER_YEARNL    365 (non-leap year)
ICMC_DAYS_PER_YEARL     366 (leap year)
DAYS_PER_YEAR(t) (t.leap_year ? ICMC_DAYS_PER_YEARL : \
                        ICMC_DAYS_PER_YEARNL)
LCMC_MS_PER_DAY         86400000L (msecs per day)
```

```
/*
 *  MACRO:  LEAP_MSEC_MOD
 *
 *  Adds or subtracts a second worth of msecs depending on the flags.
 */
#define LEAP_MSEC_MOD(t) (t.sub_leap_sec ? -1000L : t.add_leap_sec ? 1000L : 0)
```

d. csw/include/csxtct.h

```
DAYSPERYR   365
HRSPERDAY    24
```

A.3 MSW Time Variable Interfaces

The following list time-related structure used by MSW time-related interfaces. Only structures that need user's attention will be listed:

```
gettime.h:          (Popular internal time format and shared memory segment)
  typedef struct {
    short     year;          /*  Current year (defined by Set_year())
                             *  If the application is not using the TCT,
                             *  this field will default to the current
                             *  system year.
                             */
    short     days_in_year;   /*  Number of days in the year (based on
                             *  year).  Defined by Set_year()
                             */
    short     day;           /*  Day of year */
    short     hr;            /*  Hour of the day              */
    short     min;           /*  Minute of the day            */
    short     sec;           /*  Second of the day            */
    short     tenths;        /*  Tenths of second             */
    long      millisecs_of_day;
                          /*  Milliseconds of day */
    long      sec_of_day;     /*  Seconds of day               */
    long      sec_of_year;    /*  Seconds of year              */
    long      time_bias;      /*  Time bias in seconds (defined by
                             *  Set_time_bias()
                             */
    bool      valid;         /*  TRUE if time is valid.       */
    short     source;        /*  Source of TIME: FROM_TCT or FROM_SYS */
  }  TCT_TIME;

typedef struct time_chunk {
  short       mode;          /*  Timer mode:  TCT_ENABLED, TCT_DISABLED
                             */
  long        time_bias;      /*  Time bias applied to system time */
  short       sys_status;     /*  System time status */
  TCT_TIME    sys_time;       /*  Current system time.  Set by tmr task */
  short       tct_status;     /*  TCT time status */
  TCT_TIME    tct_time;       /*  Current TCT time.  Set by tmr task */
  short       year;          /*  Current year.  Set by Set_year() */
  short       days_in_year;   /*  Number of days in current year.  This
                             *  is valid only if the application has
                             *  set the year with Set_year(), otherwise,
                             *  it defaults to 365.
                             */
  long        sec_per_year;   /*  Seconds per year based on year */
} TIME_CHUNK;

          struct tct_data
          {
             unsigned char       status_byte;
             unsigned int        doy;
             long                sys_time_offset;
             int                 tct_errno;
          };

    struct time_value
    {
        char days[XTCTDLN];                 /* 3-char */
        char hours[XTCTHLN];                /* 2-char */
        char minutes[XTCTMLN];  /* 2-char */
        char seconds[XTCTSLN];    /* 2-char */
        char millisecs[XTCTMLSLN];        /* 3-char */
    };
```

```
/*   structure used to return time and day from tct_btime() */

    struct btime_value
    {
        int bdays;
        int bhours;
        int bminutes;
        int bseconds;
        int bmillisecs;
        long bmsecs_of_day;
    };

/*   structure return TCT status to user programs          */

    struct status_flags
    {
        uchar t_type;                  /* GMT or SIM */
        cbool sub_leap_sec,              /* TRUE or FALSE */
              add_leap_sec,             /* TRUE or FALSE */
              leap_year;               /* TRUE or FALSE */
     };

/*   structure used to transfer time value and status to kernel */
/*   size is 16 bytes (12 chars t_val, 4 chars t_stat) */

    struct time_struct
    {
       struct time_value t_val;
       struct status_flags t_stat;
    };

    typedef struct _DATETIME      /* date (OS/2) */
         {
         UCHAR   hours;
         UCHAR   minutes;
         UCHAR   seconds;
         UCHAR   hundredths;
         UCHAR   day;
         UCHAR   month;
         USHORT  year;
         SHORT   timezone;
         UCHAR   weekday;
         } DATETIME;

struct tctm {
  short  hour,  /* Hour */
         mins,  /* Minutes */
         secs,  /* Seconds */
         mls;   /* Milliseconds */
  };


typedef struct stat_list_node
{
```

```
/*
 *  Stat_list_node -- This is the structure used to tie monitor data
 *                    segments together into the 'spmc_MD_seg_stat_tbl.'
 *                    This table is a doubly-linked list arranged in
 *                    numerical order based upon two key items.
 *                    The key items are fields within the monitor data
 *                    segment structure 'sv_seg,' the most significant
 *                    of which is the segment source --'sv_seg.iv_seg_src.'
 *                    The next most significant is the segment ID field
 *                    'sv_seg.iv_seg_id.'  The list is arranged in
 *                    ascending order.
 */
  bool              bv_valid;         /* boolean value indicating
                                       the validity of the segment.
                                         TRUE -- segment is valid.
                                         FALSE -- segment is invalid. */
  bool              bv_changed;       /* flag indicating that the segment is
                                       in need of updating due to parameter
                                       change */
  bool              bv_locked;        /* flag indicating whether the segment
                                        has a lock placed on it for either
                                        inbound or outbound processing */
  short             iv_doy_tag;       /* Day of year tag of lock expiration */
  long              lv_mss_tag;       /* Millisecond of day tag of lock
                                         expiration */
  Lock_seg_buf      *sp_locked_seg;   /* buffer holding the locked segment and
                                        any relevant statistics */
  short             iv_timer;         /* timer counter for periodic update
                                       protocol */
  Md_seg            *sp_seg;          /* pointer to the defined segment
                                       structure. */
 struct stat_list_node *sp_next;      /* pointer to next entry in table. */
 struct stat_list_node *sp_prev;      /* pointer to previous node in table */
} Stat_list_node;

typedef struct rpt_catalog_block
{
  char        name[RPT_ID_LN];        /* report file name/id */
  long        datetime;               /* seconds of year */
  short       lines;                 /* total length */
  short       status;                /* rpt flags */
  FILE        *fp;                   /* file pointer for writing */
} RPT_CAT;
```

A.4 Date Representations:

This section lists MSW output date format:

MSW Version Date (hard-coded definition) format MM/DD/YY

parser.c:
Date mm/dd/yy format in share memory creation program (autogen).

TCT time string: DDDHHMMSSTCCR; T:status byte, CC:8-bit checksum, R:'\r'

MD Time format: DDDHHMMSSsss; right-justified and zero filled.

Dsp/Rpt Time: DDD HH:MM:SS; right-justified and zero filled.

MSW errmsg: (To CRT or log) with date in form of "DDD HH:MM:SS"

MD Segment Time format: char cv_time[13]; /* ASCII time string */

Timed Operator Directive: HHMMSS[.s] ([] indicates optional).

ACTL Log Time: DDD HH:MM:SS

Support Data Product:
NSS Table: (820-16 MON-5-206 NSS Table Format)
YY/DDD HH:MM:SS (DSS_MOD)
Time Values: TIME HH[:]MM[:]SS
YTIME DDD HH[:]MM[:]SS, where [] indicates optional entry.